streamlit_book

Sebastian Flores Benner

Aug 07, 2023

CONTENT

streamlit_book is an extension to the library streamlit, allowing an easier creation of streamlit apps with multipages and interactive activities. Jump to the introduction to see some examples or go to the documentation to see how to use it.

CHAPTER

ONE

INTRODUCTION

streamlit_book provide tools to create interactive multipaged apps on streamlit apps using python and simple folder convention.

1.1 Features

- Reads and renders python files of the provided folders.
- Multipaging compatible with native streamlit multipage. It can renders chapters (previous, next and reload buttons) or a book (collection of chapters).
- Interactive activities: true or false question, single choice question, multiple choice question and to do list.
- User answers can (optionally) be saved.
- User's token to persist a session.
- Admin view: stats on users and saved answers.

Use streamlit_book to create more rich and interactive apps, activities and quizzes!

1.2 Examples

- Happy Birds : A self contained example that mixes features of the library with a funny twist.
- The (confusion) Matrix: Take the blue pill to learn all about the confusion matrix.
- The Streamlitsaurus Rex: Will teach you to always visualize your data, and exhibits the mythical Datasaurus.
- SatSchool: interactive app how to manipulate and understand data from satellites in a range of environmental contexts.
- Template for multipaging: A template for a multipage app using streamlit_book, with public and private sections.
- stbook-methods.streamlitapp.com : All the activities and goodies in streamlit_book!
- stbook-multipaging.streamlitapp.com : How to do multipage with streamlit book!

Are you using streamlit_book? Let me know and I will add it!

CHAPTER

TWO

INSTALLATION

The repository for the code is hosted at https://github.com/sebastiandres/streamlit_book.

2.1 From pypi

You can install the library from pypi. This is the safe way. Don't stray from this path.

```
pip install streamlit_book
```

On the *requirements.txt* file it should be simply the line:

```
streamlit_book
```

Or, optionally, for the specific version X.Y.Z:

```
streamlit_book==X.Y.Z
```

2.2 From repository

You can install the library directly from the latest available version on github. This is good for testing the library, but might encounter some bugs, in which case you should let us know!

```
pip install git+https://github.com/sebastiandres/streamlit_book.git
```

On the *requirements.txt* file it should be simply the line:

```
git+https://github.com/sebastiandres/streamlit_book.git
```

CHAPTER

THREE

MULTIPAGING: A SHORT GUIDE

There are ways to have a multipage content

3.1 Native multipaging

- **Problem**: You don't want streamlit_book multipaging, but streamlit's native multipaging.
- **Solution**: Just pust your files on *pages/* and import streamlit_book on the files where you need them. You can mix streamlit_book and streamlit's native multipaging.

3.2 Single page

- **Problem**: You have one page to show, but you want to use the activities/questions provided on streamlit_book.
- **Solution**: Just import the library streamlit_book without the setup and call the required functions.

```
import streamlit as st
import streamlit_book as stb

st.set_page_config()

# No need to initialize the streamlit_book library
stb.true_or_false("Are you a robot?", False)
```

3.3 Chapter: A single document with multiple connected pages

- **Problem**: You want cute multipages, but you only need previous/next buttons.
- **Solution**: Use method *set_chapter_config* ` to set the path and other chapter configurations.

```
import streamlit as st
import streamlit_book as stb

# Set wide display
st.set_page_config()

# Set multipage
stb.set_chapter_config(path="pages/", save_answers=True)
```

Using the function *set_chapter_config*:

- Will setup the page navigation text/icons for a unique chapter.
- Will sort the python and markdown files of the given path on lexigraphic order.
- Will read and render the files into streamlit, allowing a enriched markdown/python of the implemented activities.

See the function *set_chapter_config* required and optional parameters on the *Multipaging Documentation*.

3.4 Book: Having several chapters

Requires a sidebar menu (like this demo), where each topic required a previous/next buttons.

Use *stb.set_book_config* ` to set the path and the configuration for the book.

```
import streamlit as st
import streamlit_book as stb

# Streamlit webpage properties
st.set_page_config()

# Streamit book properties
stb.set_book_config(menu_title="streamlit_book",
                    menu_icon="lightbulb",
                    options=[
                            "What's new on v0.7.0?",
                            "Core Features",
                            ],
                    paths=[
                          "pages/00_whats_new.py", # single file
                          "pages/01 Multitest", # a folder
                          ],
                    icons=[
                          "code",
                          "robot",
                          ],
                    save_answers=True,
                    )
```

Using the function *set_book_config*:

- Will setup a sidebar menu with each of the chapter.
- Each chapter is build with pagination for the provided file/folder.

The capability to render several chapters was added in version 0.7.0, and makes a direct use of an awesome library to add a sidebar menu called (streamlit_option_menu). Kudos to the creator. It delivers a professional look, and allows to add icons by name makes it a lot more user-friendly.

See the function *set_book_config* required and optional parameters on the *Multipaging Documentation*.

CHAPTER

FOUR

MULTIPAGING DOCUMENTATION

set_book_config(*options*, *paths*, *menu_title='Select a chapter'*, *menu_icon='book'*, *icons=None*, *orientation=None*, *styles=None*, *save_answers=False*, *display_page_info=True*)

Creates a book using the streamlit_option_menu library. Renders each of the corresponding chapters based on their properties. Uses the same configurations used by streamlit-option-menu and icons from bootstrap-icons.

Parameters

- **options** (*list of str*) – List of chapter names to be displayed
- **paths** (*list of str*) – List of chapter paths containging the pages (py, md) to be displayed
- **menu_title** (*str*) – Title of the menu, can be empty to be skipped.
- **menu_icon** (*str or list of str*) – Icon to be used on the menu, from bootstrap icons.
- **icons** – Icons to be used. Can be a single one used for all books, or a list of icons for each book.
- **orientation** (*str*) – Orientation of the menu. Can be "horizontal" or "vertical".
- **styles** (*dict*) – Styles to be used. See the documentation of streamlit_option_menu.
- **save_answers** (*bool*) – If True, it will save the answers in a csv file. Defaults to False.
- **display_page_info** (*bool*) – If True, it will display the page info with the name and number. Defaults to True.

Returns

None

set_chapter_config(*path='pages'*, *toc=False*, *button='top'*, *button_previous=''*, *button_next=''*, *button_refresh=''*, *on_load_header=None*, *on_load_footer=None*, *save_answers=False*, *display_page_info=True*)

Sets the book configuration, and displays the selected file.

Parameters

- **path** (*string, dict*) – The path to root directory of the the files (py or md) to be rendered as pages of the book.
- **toc** (*bool*) – If True, it will display the table of contents for the files on the path. Defaults to False.
- **button** (*str*) – "top" (default behavior) or "bottom".
- **button_previous** (*str*) – icon or text for the previous button.
- **button_next** (*str*) – icon or text for the next button.
- **button_refresh** (*str*) – icon or text for the refresh button.

- **on_load_header** (*function*) – function to be called before the page is loaded.
- **on_load_footer** (*function*) – function to be called after the page is loaded.
- **save_answers** (*bool*) – If True, it will save the answers in a csv file. Defaults to False.
- **display_page_info** (*bool*) – If True, it will display the page info with the name and number. Defaults to True.

Returns
None

CHAPTER

FIVE

INTERACTIVE ACTIVITIES

There are different interactive activities to be used in your apps:

5.1 True or False question

This functionality allows to ask a true/false type of question. It requires a question and the solution as a True/False value. Optionally, the texts for success, error and button can be customized.

true_or_false(*question*, *answer*, *success='Correct answer'*, *error='Wrong answer'*, *button='Check answer'*)

Renders a true or false question from arguments.

Parameters

- **question** (*str*) – question to be displayed before the true or false options
- **answer** (*str*) – expected answer to the question, can be True or False
- **success** (*str, optional*) – message to be displayed when the user answers correctly. If empty, no message is displayed.
- **error** (*str, optional*) – message to be displayed when the user answers incorrectly. If empty, no message is displayed.
- **button** (*str, optional*) – message to be displayed on the button that checks the answer

Returns

tuple of booleans(button_pressed, answer_correct) with the button status and correctness of answer

Return type

tuple of bool

5.2 Single choice question

This functionality allows to ask a single choice question. It requires a question, the alternatives and the (unique) answer. Optionally, the texts for success, error and button can be customized.

single_choice(*question*, *options*, *answer_index*, *success='Correct answer'*, *error='Wrong answer'*, *button='Check answer'*)

Renders a single-choice question from arguments.

Parameters

- **question** (*str*) – question to be displayed before the single-choice options

- **options** (*str*) – list of options to be displayed.
- **answer** (*int*) – index (starting at 0) of the expected answer to the question
- **success** (*str, optional*) – message to be displayed when the user answers correctly
- **error** (*str, optional*) – message to be displayed when the user answers incorrectly
- **button** (*str, optional*) – message to be displayed on the button that checks the answer

Returns
tuple of booleans with button press status and correctness of answer

Return type
tuple of bool

5.3 Multiple choice question

This functionality allows to ask a multiple choice question. It requires a question, the alternatives and the answer(s). Optionally, the texts for success, error and button can be customized.

multiple_choice(*question*, *options_dict*, *success='Correct answer'*, *error='Wrong answer'*, *button='Check answer'*)

Render a multiple choice question from the given parameters.

Parameters

- **question** (*str*) – question to be displayed before the multiple-choice options
- **options_dict** – dictionary of options to be displayed, with the option text as key and the boolean answer as value
- **answer** (*int*) – index (starting at 0) of the expected answer to the question
- **success** (*str, optional*) – message to be displayed when the user answers correctly
- **error** (*str, optional*) – message to be displayed when the user answers incorrectly
- **button** (*str, optional*) – message to be displayed on the button that checks the answer

Returns
tuple of booleans with button press status and correctness of answer

Return type
tuple of bool

5.4 To do list

This functionality allows to create to-do list. It only has as optional argument the text for success.

to_do_list(*tasks*, *header=''*, *success='Bravo!'*)

Renders the tasks as a to-do list, with optional header and success message. The tasks are a dictionary of tasks (supposed to be ordered as Python +3.6) with their completed (True) or to-do (False) status as a checkbox.

Parameters

- **tasks** (*dict*) – dictionary of tasks in format string:bool
- **header** (*str, optional*) – description of the tasks
- **success** (*str, optional*) – success message

Returns
boolean with the exit status of the function

Return type
bool

CHAPTER

SIX

OTHER GOODIES

But wait, there's more!

6.1 Colored Expanders

More colors to your expander!!! You just have to start the expander header with the right keyword.

6.2 Echo

The same old echo function from streamlit, but with an additional boolean parameter to control whether the output is shown or not. Intended to be paired with a checkbox, as shown on the example.

echo(*code_location='above'*, *show=True*)

Whether to show the echoed code before or after the results of the executed code block. :param code_location: "above" or "below" :type lines: str :param show: Boolean to show or hide the code block :type lines: bool :return: None :rtype: none Copied and improved from Streamlit's github

Interactive example:

6.3 Share on Social Media

You can create share buttons for social media. This is based on the code I found at https://sharingbuttons.io/, which after a couple of hours of googling turned out to be the simplest way to share the content. Kudos to creator. It has some minor modifications, but it's mostly a wrapping of the code for stremlit.

It requires as arguments the message and the url.

share(*my_text*, *my_url*)

This function takes a url and a text and displays clickable sharing buttons in html.

Parameters

- **my_text** (*str*) – the text to share on social media
- **my_url** (*str*) – the url to share on social media

CHAPTER

SEVEN

ROADMAP

This is the roadmap for the next version of the library.

- Preserving answers to questions - it currently resets the answers to the default values, even if answers have been provided/saved.
- Setting a limit number of answers to a question - it currently allows unlimited answers.
- Reimplement the Table of Contents.
- Save when a user has accepted (do not repeat the dismiss).
- More activities: Text entry, Code entry, File upload, …

INDEX